**Part 3 - Detailed Design**

**Architecture-**

The architecture best suits our project is **MVC**.

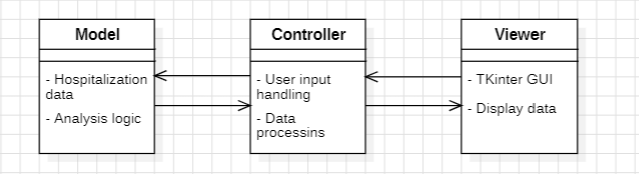
**MVC Architecture in our project:**

* **Model**:
  + Manages hospitalizations and patient data, trained ML data, and other relevant information.
  + Handles data retrieval, storage, and processing.
* **View**:
  + GUI created using Tkinter.
  + Presents data to hospital staff.
  + Accepts user input.
* **Controller:**
  + Acts as a bridge between the Model and the View, mostly automated by Tkinter.
  + Handles user input, processes it, and updates the Model and View accordingly.

**Data Storage:**

* The hospitalization list and data on patients are stored in CSV.
* Trained ML models will be stored in files.
* Other relevant data is stored in separate files.

**Graphic Description**



**data description:**

* **Hospitalizations list** - a CSV file that has the following fields:
  + serial number
  + age
  + department
  + hospitalization duration
  + date and time of release of the patient
  + diagnose of discharge from the first hospitalization
  + Time (in days) from the first discharge to the next hospitalization
  + department
  + date and time of the second hospitalization
  + duration time in ER
  + Reason for contacting triage
  + Diagnoses in the emergency room
  + Hospitalist (doctor)
  + Diagnoses at admission for re-hospitalization
* **ER rates list** - a CSV file that has the following fields:
  + date of arrival
  + day of the week
  + walk-in clinic ER
  + internal ER
  + Infectious emergency medicine
* **Diagnose list** - a CSV file that contains all possible diagnoses for hospitalization.
* **Doctors list -** a CSV file that contains the following fields:
  + the doctor's name
  + level of experience (intern / mid-level / senior)
  + ER doctor / internal department doctor

**API specification:**

User interface:

* Upload CSV of readmission hospitalization data.
* Upload CSV of ER rates data.
* Upload CSV of diagnoses data.
* Upload CSV of data of doctors.
* choose a parameter for the graph generator - for one parameter graphs.
* choose parameters for the graph generator - for two parameter graphs.
* Show a graph with one parameter.
* Show a graph with two parameters.
* Save graphs as jpeg.
* Choose an algorithm for training ML.
* Prediction results of the ML model display.
* Save an ML model.

**interface design:**

****

**Programming Languages and Tools** -

The reason we selected Python is that we need to make a statistics analysis. For that reason, we chose to work with Matplotlib, Pandas, and Numpy libraries for statistics and graph generators and SKlearn for the machine learning model.

Also, we wanted a reactive UI that could be used with the TkInter library.

**Algorithm description**

**Machine Learning Algorithms**

**1. Linear Regression:**

***Algorithm:*** *Linear regression* is a supervised learning algorithm used for regression tasks where the goal is to predict a continuous value based on input features. It fits a linear model to the data by minimizing the sum of squared differences between the observed and predicted values.

**Time Complexity:** The time complexity of linear regression mainly depends on the method used for optimization.

For ordinary least squares (OLS), a standard linear regression method, the time complexity is typically O(n^2 \* m), where 'n' is the number of features and 'm' is the number of data points.

**2. Decision Trees:**

**Algorithm:** Decision trees are a supervised learning algorithm for classification and regression tasks. They recursively split the data into subsets based on the feature that best separates the data. Each split is chosen to maximize the information gain (for classification) or minimize the impurity (for regression).

**Time Complexity:** The time complexity of decision trees depends on the number of data points' m', the number of features' n', and the depth of the tree 'd'. Building a decision tree typically has a time complexity of O(n \* m \* log(m)) or O(n \* m^2) for standard algorithms like CART (Classification and Regression Trees). However, decision tree algorithms can be optimized and parallelized, leading to faster training times in practice. Additionally, predicting with a decision tree typically has a time complexity of O(log(m)).

**3. Support Vector Machine (SVM):**

**Algorithm:** They are used for classification and regression tasks.

In classification, SVM finds the hyperplane that best separates the classes in feature space while maximizing the margin between the classes.

In regression, SVM finds the hyperplane that best fits the data while minimizing deviations from the hyperplane.

**Time Complexity:** of SVM depends on the chosen kernel and the algorithm used for optimization.

For linear SVM, where no kernel trick is applied, the time complexity typically ranges from O(n^2 \* m) to O(n^3 \* m), where 'n' is the number of features and 'm' is the number of data points.

For non-linear SVM, especially with kernels like polynomial or Gaussian, the time complexity can be higher, potentially reaching O(m^2 \* n^2).

However, various optimization techniques and algorithms have been developed to reduce the computational cost, making SVM practical for large datasets.

**4. k-Nearest Neighbors (KNN):**

**Algorithm:** KNN is a supervised learning algorithm for classification and regression tasks. It makes predictions based on the majority class (for classification) or the average value (for regression) of the k nearest data points in the feature space.

**Time Complexity:** The time complexity of the KNN algorithm can be divided into two main parts: training and prediction.

- Training: For KNN, there is no explicit training process as the algorithm stores all the training data - O(1).

- Prediction: The time complexity of prediction in KNN depends on the number of data points' m', the number of features' n', and the value of 'k.'

For each prediction, the algorithm must compute the distances between the query point and all the training data points - O(m \* n), where 'm' is the number of data points and 'n' is the number of features.

However, with efficient data structures like KD-trees or Ball-trees, the prediction time complexity can be reduced to O(log(m) \* n) or even O(log(m) \* k) for finding the k nearest neighbors.